An Eprints Apache Log Filter for Non-Redundant Document Downloads by Browser Agents

Ed Sponsler
Caltech Library System
http://resolver.caltech.edu/CaltechLIB:SPOeal04

December, 2004

Contents

1 Abstract

Web log files record a vast amount of information and much of it just gets in the way of meaningful observational studies on usage. It is therefore necessary to filter out the junk in a deliberate way before making statements on how the web is being used.

This report describes the methods and scripts used to accomplish apache web log filtering and report generation. It is open to scrutiny and freely available for others to use.

These methods were used to generate reports presented at the following conferences. Attendees at these conferences may be curious to know exactly how the data were generated, and that is the purpose of this report.

Doglas, Kim. (2004) Keynote: Experiences and Challenges. International Conference on Developing Digital Institutional Repositories, Hong Kong.

Van de Velde, Eric. (2004) CODA, an Eprints Open Digital Archive. International Conference on Developing Digital Institutional Repositories, Hong Kong.

Sponsler, Ed. (2004) Caltech ETD Collection Analysis: Who Accesses What and Why? The 7th International Symposium on Electronic Theses and Dissertations, U. of Kentucky.

2 Processing Apache Log Files

This study analyzes Apache 'combined' format log files generated by various versions of Eprints[1] and Virginia Tech ETD-db[2] software.

2.1 Human vs. Robot Access

For the purposes of these observational studies, only log entries generated by web browsers are interesting.

Known web browsers may be identified by the agent identifier recorded in the apache log. Several resources on the web catalog Agent IDs[3]. The ZyTrax source is useful since each browser Agent ID is explained in detail, giving us some confidence that these strings indicate browsers and not robots.

Once a list of known browser Agent IDs is prepared it may be used by a script to cull the log entries matching one of the known browser agent identifiers.

2.1.1 Scripts

The browser Agent ID list is prepared by extracting the identifier strings from the other text in the html file, outputting these identifiers to a text file. The utility wget is used to obtain the source html from ZyTrax.

The filter script parse-browser-ids.pl goes through the trouble of sorting the browser strings so that artifacts caused by human error in the html code float to the top. Examples of these artifacts are strings which begin with a space or contain misplaced tags. These are easy enough to edit by hand prior to using the list by another script to parse a log file.

The log parsing script split-human-robot.pl produces three output files: log entries which match a browser agent ID (access-log-human), those which don't (access-log-robot) and those which do not contain an Agent ID (access-log-no-agent). In this way all of the original log entries may be found in one of these three files.

The script also sends access-log-human to stdout to support piping the data to another script.

```
Usage:
$ wget http://www.zytrax.com/tech/web/browser_ids.htm
$ ./parse-browser-ids.pl < browser_ids.htm > browser-ids.dat
[Fix possible artifacts in browser-ids.dat]
```

Script: **parse-browser-ids.pl**

[1] http://eprints.org

[2] http://scholar.lib.vt.edu/ETD-db/developer/download/

[3] See ZyTrax: http://www.zytrax.com/tech/web/browser_ids.htm; PGTS: http://www.pgts.com.au/pgtsj/pgtsj0212d.html; and Psychedelix: http://www.psychedelix.com/agents.html

```
#!/usr/bin/perl -w
my (%br_strs, $br_str);
while (<>) {$br_strs{$1}=0 if (m~g-c-[sn]">(.*?)</p>~)}
foreach $br_str (sort keys %br_strs) {print "$br_str\n";}
```

Script: **split-human-robot.pl**

```
#!/usr/bin/perl -w
open (ROBOT, ">access-log-robot") || die;
open (HUMAN, ">access-log-human") || die;
open (ERROR, ">access-log-noagent") || die;
open (IDS, "<browser-ids.dat") || die;
my (%ids, $id); my $match=0;
while (<IDS>) {chomp; $ids{$_}=1;}
while (<>) {
        if (/.+\s"(.+)"$/) {
                foreach $id (keys %ids) {$match=1 if ($id eq $1);}
        }else{ print ERROR; }
        if ($match) {print; print HUMAN; $match=0;} else {print ROBOT;}
}
```

3 Processing Algorithm

The following algorithm is used after the logs have been filtered of robots. The main script also filters the logs based on the other parameters described in this section.

```
1) Read the next line of the log file
2) Store the IP number if it is the first time we have seen it,
otherwise goto 4). This list of IPs is used to generate Geographical
Analysis.
3) Extract the month/year from the log and add one to a counter tracking
the number of times step 2) has been true for this month. This data
is used in the Usage Chart to show the number of first time visitors.
4) Extract the record identifier and IP. If this combination has never
been seen before then store the combination as well as the log line
itself, otherwise goto 1). The number of IPs associated with the same
record identifier is calculated and provides the range (y axis) in the
Scatter Plot. The log lines collected at this step become the raw
material used by AWstats to produce the web stats pages.
5) Extract the month/year and add one to the counter tracking the number
of times step 4) has been true for this month. This data is used to
create downloads/month in the Usage Chart.
6) Goto 1)
```

3.1 Document File Format

Document file types of interest are isolated from the logs by matching filenames with known file format extensions such as .pdf and ps. Files having other extensions such as .htm, .gif or .png are ignored.

3.2 Successful Downloads

Log entries are discriminated based on http status code. Successful codes are accepted for analysis, others are ignored. Accepted codes are: 200 (OK), 206 (partial content) and 304 (not modified). All others are ignored, such as: 403 (forbidden), 404 (file not found) and 500 (internal server error)[4].

3.3 Removing Redundancy

A redundant download is defined as a host accessing the same record more than once. It is possible to remove these from the log file by storing the set of host IPs that have accessed each record and rejecting subsequent log entries whose source IP matches one in the list.

This process is important since it removes uninteresting log entries from analysis. There are many ways redundancy is introduced in log files. It is common for a web server to chop a large document up into smaller chunks prior to downloading to the browser. When this happens a single download event appears misleadingly as multiple downloads in the log file.

There may also be cases in which a robot agent masquerades its identifier as one on the known browser list or a user may return to the server multiple times to download the same document rather than store a local copy.

Removing redundancies is a simple way to control for these confounding influences.

3.4 Obtaining Document Metadata

A scatter plot of document downloads over submission date provides a qualitative picture of the archive's activity. The submission date is obtained by querying the archive database using the identifier parsed from the apache log. It is useful to obtain other record metadata as well, such as title and URL.

3.5 Preparing Output for Web Analysis Software

One of the files produced by the following script is a chronologically ordered, filtered log file for processing by other software such as AWstats[5].

3.6 Script I/O

The script takes an Apache log in combined format as input. After filtering, it outputs: 1) first time accesses in apache log format to NON_REDUNDANT_LOGS; 2) redundant accesses to REDUNDANT_LOGS; 3) logs which don't match a legitimate access regular expression to ILLEGITIMATE_LOG; 4) a table of the number of non-redundant accesses per document, including the document's submission date and title to COUNT_RECORD; 5) a table of the number of accesses and number of new visitors per month to COUNT_MONTHLY; 6) a complete list of ip numbers that have downloaded at least one document to ALL_IPS; 7) a list of record identifiers missing from the archive database to MISSING_ID.

The script ensures that every log line, unexpected regular expression mismatches and missing record identifiers end up in appropriate output files.

3.6.1 Script

This is the main perl script[6]

Script: **filter-apache.pl**

[4] More info on http status codes: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[5] http://awstats.sourceforge.net/

[6] Download: http://resolver.caltech.edu/CaltechLIB:SPOeal04

```
#!/usr/bin/perl -w
# filter-apache.pl
# by Ed Sponsler, December, 2004
# eds@library.caltech.edu

use Time::Local;
use DBI;

##########################
# BEGIN Configuration #
##########################

my $db_name = 'databasename';
my $db_user = 'user';
my $db_pass = 'password';
open (NON_REDUNDANT_LOGS, ">access-log-non-redundant");
open (REDUNDANT_LOGS, ">access-log-redundant");
open (ILLEGITIMATE_LOGS,">access-log-illegitimate");
open (ALL_IPS, ">all-ips.dat");
open (COUNT_MONTHLY, ">count-monthly.dat");
open (COUNT_RECORD, ">count-record.dat");
open (MISSING_ID, ">count-record.err");

# Regular Expression Numbered Variables
# $1 = IP
# $2 = day
# $3 = month
# $4 = year
# $5 = hour
# $6 = minute
# $7 = second
# $8 = record id

#############################################################################
# NOTE: Each regular expression should be one long line! They have been
# typeset as shown in order to fit on the page. You will need to join
# the lines making one long line per regular expression assignment prior
# to running the script.
#############################################################################

my $re_1 =
qr~^(.+?)\s.+\[(\d+)/(\w+)/(\d+):(\d+):(\d+):(\d+)\s.+/archive/0*
([1-9]\d*)/\d+/.+?\.(pdf|ps|PDF|PS)\s.+?\s(200|206|304)\s.+$~;

my $re_2 =
qr~^(.+?)\s.+\[(\d+)/(\w+)/(\d+):(\d+):(\d+):(\d+)\s.+/\d+/\d+/\d+/\d+
/0*([1-9]\d*)-\d+/.+?\.(pdf|ps|PDF|PS)\s.+?\s(200|206|304)\s.+$~;
```

```
my $re_3 =
qr~^(.+?)\s.+\[(\d+)/(\w+)/(\d+):(\d+):(\d+):(\d+)\s.+GET\s/(\d+)/\d+
/.+?\.(pdf|ps|PDF|PS)\s.+?\s(200|206|304)\s.+$~;

######################
# END Configuration #
######################

my %mon_1 = (Jan=>'0',Feb=>'1',Mar=>'2',Apr=>'3',May=>'4',Jun=>'5',
Jul=>'6',Aug=>'7',Sep=>'8',Oct=>'9',Nov=>'10',Dec=>'11');

my %mon_2 = (Jan=>'01',Feb=>'02',Mar=>'03',Apr=>'04',May=>'05',
Jun=>'06',Jul=>'07',Aug=>'08',Sep=>'09',Oct=>'10',Nov=>'11',Dec=>'12');

#
# Process Each Line of the Apache Log File
#

while (<>) {
        if (($_ =~ $re_1)||($_ =~ $re_2)||($_ =~ $re_3)) {
                $month="$4-$mon_2{$3}-01";
                $months_represented{$month} = 1;
# count non-redundant archive visits per month
                if (!exists $all_ips{$1}) {
                        $all_ips{$1}=1;
                        if (exists $count_non_redundant_visits{$month}) {
                                $count_non_redundant_visits{$month} += 1;
                        }else{$count_non_redundant_visits{$month} = 1;}
                }
# store non-redundant record access logs
                if (!exists $non_redundant_record_accesses{$8}{$1}) {
                        $non_redundant_record_accesses{$8}{$1}=1;
                        $epoch_secs = timelocal($7,$6,$5,$2,$mon_1{$3},$4);
                        $logs{$epoch_secs} = $_;
# and then count non-redundant record accesses per month
                        if (exists $count_non_redundant_accesses{$month}) {
                                $count_non_redundant_accesses{$month} += 1;
                        }else{$count_non_redundant_accesses{$month} = 1;}
                }else{print REDUNDANT_LOGS;}
        }else{print ILLEGITIMATE_LOGS;}
}

#
# Output Reports
#

# monthly totals
$accesses_sum = 0;
```

```
$visits_sum = 0;
print COUNT_MONTHLY "MONTH\tACCESSES\tACCESSES_SUM\tVISITS\tVISITS_SUM\n";
foreach $month (sort keys %months_represented) {
        $accesses_sum += $count_non_redundant_accesses{$month};
        $visits_sum += $count_non_redundant_visits{$month};
        print COUNT_MONTHLY "$month\t$count_non_redundant_accesses{$month}\t"
        . "$accesses_sum\t$count_non_redundant_visits{$month}\t$visits_sum\n";
}
# a list of all the IPs
foreach $ip (sort keys %all_ips) {print ALL_IPS "$ip\n";}
# non-redundant logs for analysis by AWstats, webalyzer, etc.
foreach $epoch_secs (sort keys %logs) {
        print NON_REDUNDANT_LOGS $logs{$epoch_secs};
}
# record totals with title and submission date
$dsn = "DBI:mysql:database=$db_name";
$dbh = DBI->connect($dsn, $db_user, $db_pass, {RaiseError => 1});
print COUNT_RECORD "UID\tSUB_DATE\tDOWNLOADS\tTITLE\n";
foreach $id (sort keys %non_redundant_record_accesses) {
        $count_ips = scalar(keys %{$non_redundant_record_accesses{$id}});
        $sql = "SELECT datestamp,title from archive where eprintid='$id'";
        $sth = $dbh->prepare($sql); $sth->execute;
        $sth->bind_columns(\($sdate, $title));
        if ($sth->fetch) {
                print COUNT_RECORD "$id\t$sdate\t$count_ips\t$title\n";
        }else{ print MISSING_ID "Record ID: $id not found!\n"; }
}
```

3.7 Shell Script for Batch Processing

In this example, Apache logs are rotated weekly into a special directory and gziped. The directory name is determined by a short string representing the archive, such as 'cstr' for the Computer Science Technical Reports archive. Given this string as its only argument, the shell script knows where to find the Apache logs for that archive and what to name the various output files.

The shell variable DATA stores the path where you would like to store the output files and LIB is the path to the scripts. Be sure split-human-robot.pl is able to find browser-ids.dat.

```
Usage:
$ ./get_log_proc.sh cstr
```

```
#!/bin/bash
# get_log_proc.sh
# Shell script for processing apache log files
DATA=/home/eprints/proj_coda_analysis/current/log_proc/output
LIB=/home/eprints/proj_coda_analysis/current/log_proc/lib
        find /var/log/httpd/$1/archive/ -name 'access_log*' \
                |xargs gunzip -c|$LIB/split-human-robot.nov2004.pl \
                |$LIB/filter-apache.nov2004.pl
```

```
mv access-log-human $DATA/$1.human
mv access-log-robot $DATA/$1.robot
mv access-log-noagent $DATA/$1.no-agent
mv access-log-illegitimate $DATA/$1.illegitimate
mv access-log-non-redundant $DATA/$1.non-redundant
mv access-log-redundant $DATA/$1.redundant
mv all-ips.dat $DATA/$1.all-ips
mv count-monthly.dat $DATA/$1.count-monthly
mv count-record.dat $DATA/$1.count-record
mv count-record.err $DATA/$1.count-record.err
```

4 Generating Statistical Reports

At this point we have various output files useful in generating charts or further analysis by other applications such as AWstats. The underlying theme for all the data is to count only document downloads by humans (web browsers) and in which any host IP may download a particular record only once.

4.1 Accesses vs. Submission Date Scatter Plot

In this view, the number of accesses per document is plotted over the date of submission to observe the archives activity. To generate this report, import the data from the count-record.dat file into Excel and create a scatter plot using the submission date as the x-axis and access count per record as the y-axis.

4.2 Growth per Month

How many documents are submitted per month into each archive? This data is contained in count-monthly.dat.

4.3 AWstats and other Web Analysis Software

Web analysis software packages such as AWstats may use any of the various apache log output files: access-log-human, access-log-robot or access-log-non-redundant.

4.4 Geographical Analysis

The all-ips.dat file is useful if your are interested in knowing where your users are coming from. A neat utility called geoip-lookup is freely available as perl module[7].

This database associates IPs number ranges with the registrant, such as which country the IP registrant is located. Once the perl module is installed and database is downloaded, the following simple script will turn your list of IP numbers (all-ips.dat) into a table of number of accesses per country.

```
#!/usr/bin/perl -w
while(<>) {
        $country = `geoip-lookup -l $_`;
        chomp ($country);
        if (exists $count_ip{$country}) {$count_ip{$country}+=1;
                }else{$count_ip{$country}=1;}
}
```

[7] http://www.maxmind.com/app/perl

```
foreach $country (sort keys %count_ip){
        print "$country\t$count_ip{$country}\n";
}
```

4.5 Archive Growth

Another way to look at archive activity is to chart the number of deposits made per month. This has nothing to do with Apache log files, but can be used together with these other reports.

The following script simply looks up all of the datestamp values in each of the supplied eprints databases and produces a report of the number of deposits per month.

It takes no input and outputs to stdout. You will of course need to supply the eprints database names, user and password.

```
#!/usr/bin/perl -w
use DBI;
my @dbs = ('eprints-database1','eprints-database2');
my $db_user = 'user';
my $db_pass = 'password';
foreach $db (@dbs) {
        $dsn = "DBI:mysql:database=$db";
        $dbh = DBI->connect($dsn, $db_user, $db_pass, {RaiseError => 1});
        $sql = 'select datestamp from archive';
        $sth = $dbh->prepare($sql); $sth->execute;
        $sth->bind_columns(\($sdate));
        while ($sth->fetch) {
                if ($sdate =~ /(\d+-\d+)-\d+/) {
                        if (exists $dat{$db}{$1}) {$dat{$db}{$1} += 1;}
                                else {$dat{$db}{$1}=1;}
                }
        }
}

foreach $sdate (sort keys %sdates) {print "${sdate}-01\t$sdates{$sdate}\n";}
foreach $db (sort keys %dat) {print "\t$db";}
foreach $db (sort keys %dat) {
        foreach $sdate (sort keys %{$dat{$db}}) {$dates{$sdate}=1;}
}
foreach $date (sort keys %dates) {print "\n$date";
        foreach $db (sort keys %dat) {
                if (exists $dat{$db}{$date}) {print "\t$dat{$db}{$date}";}
                        else {print "\t0";}
        }
}
print "\n";
```

5 Quality Control

It is important to survey the output files to fully appreciate what the scripts are doing. Every possible outcome is accounted for in the output files. Every log file inputed will find a home in one of the various

output log files. All mismatch errors and errors in sql database lookups end up in one file or another. After you run the scripts, glance over all of the output files to discern the legitimacy of this method to accurately describe the observational reports.